
Subject: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)

Posted by [serue](#) on Wed, 29 Aug 2007 20:05:15 GMT

[View Forum Message](#) <> [Reply to Message](#)

>From aec05999084bf3a94add66e98462652ed9408f86 Mon Sep 17 00:00:00 2001

From: sergeh@us.ibm.com <sergeh@us.ibm.com>

Date: Wed, 22 Aug 2007 15:03:57 -0700

Subject: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)

Introduce sys_hijack (for x86 only). It is like clone, but in place of a stack pointer (which is assumed null) it accepts a pid. The process identified by that pid is the one which is actually cloned. Some state - include the file table, the signals and sighand (and hence tty), and the ->parent are taken from the calling process.

The effect is a sort of namespace enter. The following program uses sys_hijack to 'enter' all namespaces of the specified pid. For instance in one terminal, do

```
hostname
qemu
ns_exec -u /bin/sh
hostname ab
echo $$
1073
```

In another terminal then do

```
hostname
qemu
hijack 1073
hostname
ab
```

Changelog:

Aug 23: send a stop signal to the hijacked process
(like ptrace does).

```
=====
hijack.c
=====
```

```
int do_clone_task(void)
{
    execl("/bin/sh", "/bin/sh", NULL);
}
```

```

int main(int argc, char *argv[])
{
    int pid;
    int ret;
    int status;

    if (argc < 2)
        return 1;
    pid = atoi(argv[1]);

    ret = syscall(327, SIGCHLD, pid, NULL, NULL);

    if (ret == 0) {
        return do_clone_task();
    } else if (ret < 0) {
        perror("sys_hijack");
    } else {
        printf("waiting on cloned process %d\n", ret);
        ret = waitpid(ret, &status, __WALL);
        printf("cloned process exited with %d (waitpid ret %d)\n",
            status, ret);
    }

    return ret;
}
=====

```

Signed-off-by: sergeh@us.ibm.com <hallyn@kernel.(none)>

```

arch/i386/kernel/process.c      | 51 ++++++
arch/i386/kernel/syscall_table.S |  1 +
include/asm-i386/unistd.h       |  3 +-
include/linux/ptrace.h          |  2 +
include/linux/sched.h           |  1 +
include/linux/syscalls.h        |  1 +
kernel/ptrace.c                 | 10 ++++++
7 files changed, 68 insertions(+), 1 deletions(-)

```

```

diff --git a/arch/i386/kernel/process.c b/arch/i386/kernel/process.c
index e01ddac..b13de30 100644

```

```

--- a/arch/i386/kernel/process.c

```

```

+++ b/arch/i386/kernel/process.c

```

```

@@ -789,6 +789,57 @@ asmlinkage int sys_clone(struct pt_regs regs)
    return do_fork(clone_flags, newsp, &regs, 0, parent_tidptr, child_tidptr);
}

```

```

+asmlinkage int sys_hijack(struct pt_regs regs)
+{

```

```

+ unsigned long clone_flags;
+ int __user *parent_tidptr, *child_tidptr;
+ pid_t pid;
+ struct task_struct *task;
+ int ret = -EINVAL;
+
+ clone_flags = regs.ebx;
+ pid = regs.ecx;
+ parent_tidptr = (int __user *)regs.edx;
+ child_tidptr = (int __user *)regs.edi;
+
+ rcu_read_lock();
+ task = find_task_by_pid_type(PIDTYPE_PID, pid);
+ if (task)
+ task_lock(task);
+ rcu_read_unlock();
+
+ /* Serge: I'm not clear on this. Do I need to grab a write
+  * lock to tasklist_lock, like ptrace does? */
+
+ if (task) {
+ if (!ptrace_may_attach_locked(task)) {
+ ret = -EPERM;
+ goto out_put_task;
+ }
+ if (task->ptrace || (task->state & PF_EXITING)) {
+ ret = -EBUSY;
+ goto out_put_task;
+ }
+ task->ptrace |= PT_HIJACKED;
+ force_sig_specific(SIGSTOP, task);
+ task_unlock(task);
+
+ ret = do_fork_task(task, clone_flags, regs.esp, &regs, 0,
+ parent_tidptr, child_tidptr);
+
+ task_lock(task);
+ task->ptrace = 0;
+ task_unlock(task);
+ wake_up_process(task);
+ task = NULL;
+ }
+
+out_put_task:
+ if (task)
+ task_unlock(task);
+ return ret;
+}

```

```

+
/*
 * This is trivial, and on the face of it looks like it
 * could equally well be done in user mode.
diff --git a/arch/i386/kernel/syscall_table.S b/arch/i386/kernel/syscall_table.S
index df6e41e..495930c 100644
--- a/arch/i386/kernel/syscall_table.S
+++ b/arch/i386/kernel/syscall_table.S
@@ -326,3 +326,4 @@ ENTRY(sys_call_table)
 .long sys_fallocate
 .long sys_revokeat /* 325 */
 .long sys_frevoke
+ .long sys_hijack
diff --git a/include/asm-i386/unistd.h b/include/asm-i386/unistd.h
index 006c1b3..fe6eeb4 100644
--- a/include/asm-i386/unistd.h
+++ b/include/asm-i386/unistd.h
@@ -332,10 +332,11 @@
#define __NR_fallocate 324
#define __NR_revokeat 325
#define __NR_frevoke 326
+#define __NR_hijack 327

#ifdef __KERNEL__

-#define NR_syscalls 327
+#define NR_syscalls 328

#define __ARCH_WANT_IPC_PARSE_VERSION
#define __ARCH_WANT_OLD_READDIR
diff --git a/include/linux/ptrace.h b/include/linux/ptrace.h
index ae8146a..ca953ab 100644
--- a/include/linux/ptrace.h
+++ b/include/linux/ptrace.h
@@ -68,6 +68,7 @@
#define PT_TRACE_VFORK_DONE 0x00000100
#define PT_TRACE_EXIT 0x00000200
#define PT_ATTACHED 0x00000400 /* parent != real_parent */
+#define PT_HIJACKED 0x00000800 /* not ptrace, but hijack ongoing */

#define PT_TRACE_MASK 0x000003f4

@@ -97,6 +98,7 @@ extern void __ptrace_link(struct task_struct *child,
extern void __ptrace_unlink(struct task_struct *child);
extern void ptrace_untrace(struct task_struct *child);
extern int ptrace_may_attach(struct task_struct *task);
+extern int ptrace_may_attach_locked(struct task_struct *task);

```

```

static inline void ptrace_link(struct task_struct *child,
                               struct task_struct *new_parent)
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 7fa6710..b46ae34 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1534,6 +1534,7 @@ extern int allow_signal(int);
extern int disallow_signal(int);

extern int do_execve(char *, char __user * __user *, char __user * __user *, struct pt_regs *);
+extern long do_fork_task(struct task_struct *task, unsigned long, unsigned long, struct pt_regs *,
unsigned long, int __user *, int __user *);
extern long do_fork(unsigned long, unsigned long, struct pt_regs *, unsigned long, int __user *,
int __user *);
struct task_struct *fork_idle(int);

diff --git a/include/linux/syscalls.h b/include/linux/syscalls.h
index f696874..5bc7384 100644
--- a/include/linux/syscalls.h
+++ b/include/linux/syscalls.h
@@ -616,5 +616,6 @@ int kernel_execve(const char *filename, char *const argv[], char *const
envp[]);

asmlinkage long sys_revoket(int dfd, const char __user *filename);
asmlinkage long sys_frevoket(unsigned int fd);
+asmlinkage long sys_hijack(unsigned long flags, pid_t pid, int __user *ptid, int __user *ctid);

#endif
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index 085943d..4f5c6a0 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -158,6 +158,13 @@ int ptrace_may_attach(struct task_struct *task)
return !err;
}

+int ptrace_may_attach_locked(struct task_struct *task)
+{
+ int err;
+ err = may_attach(task);
+ return !err;
+}
+
int ptrace_attach(struct task_struct *task)
{
int retval;
@@ -195,6 +202,9 @@ repeat:
/* the same process cannot be attached many times */

```

```
if (task->ptrace & PT_PTRACED)
    goto bad;
+ /* ... or ptraced while being hijacked */
+ if (task->ptrace & PT_HIJACKED)
+     goto bad;
    retval = may_attach(task);
    if (retval)
        goto bad;
--
1.5.1
```

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
